EE 3170 Microcontroller Applications

Lecture 5: Instruction Subset & Machine Language:
Introduction to the Motorola 68HC11
- Miller §2.1 & §2.2

Based on slides for ECE3170 by Profs. Davis, Kieckhafer, Tan, and Cischke

Perspective on Programming

- High-Level languages (e.g. C, C++, Java)
  - hide processor details from programmer
  - provide a common “abstract” programming model
  - allow code to be ported between processors

- Most microcontroller applications
  - require detailed operation of hardware components
  - require intimate understanding of the hardware
  - require detailed control of program and I/O timing
  - can not be done entirely in an abstract high-level language
  - Some assembly language – drivers

Processor Programming Model

- To exploit a processor, you must first understand:
  - The processor register organization
  - The memory organization
  - The instruction formats
  - The basic microprocessor operation cycle

- Hardware & software are not separate topics for microcomputer
  - Programming can not be done well w/o good understanding of registers in microprocessors & the memory

Lecture Overview

- Perspective on Programming
- Register Organization Issues
  - 68HC11 Register Organization
- Memory Organization Issues
  - 68HC11 Memory Model
- 68HC11 Instruction formats
- Instructions in memory
- 68HC11 fetch/execute cycle
- Detailed fetch/execute example
  - 3-byte instruction cycle
  - loading number from memory into accumulator
How Do We Manipulate Data in Computers?

- We usually manipulate data in **registers** in the processing unit.
  - Manipulate means:
    - adding numbers
    - complementing a number
    - shifting a number
  - Registers are groups of flip-flops.
- Some computers can manipulate data in **memory**.

Register Organization Options

- **Number of Registers**:
  - More registers:
    - Greater Capacity
    - More On-Chip Real-Estate
    - Larger Operand Fields in Instruction
  - Fewer registers:
    - More "spilling" of program data to memory space
    - Smaller Operand fields in instruction
  - Observation: few routines employ more than 32 regs.
  - Result: 32 regs became quasi-standard (by 1990s)

Register Organization Options

- **Register Uniformity**:
  - Degree to which registers can be used interchangeably
  - **One Extreme**: All General Purpose Registers (GPR's)
    - Registers can be used interchangeably
    - Greater flexibility for programmer
    - Longer Instructions (explicitly specify each register)
  - **Other Extreme**: All Special Purpose Registers
    - e.g.: accumulator, stack pointer, index register
    - Reduces flexibility for programmer: Shorter Instruction (regs are implicit in the opcode)

68HC11 Register Organization

- **Small number of special purpose registers**
  - Saves on-chip real estate
  - Shortens the instructions (saves memory)
  - 8-bit accumulators
  - 16-bit accumulator
  - Index register
  - Index register
  - Stack pointer
  - Program counter
  - Condition code register

Figure 2-1 Motorola 68HC11 Programming Model
68HC11 Register Organization

- **Accumulators** -- data registers
  - A and B – 8 bits each
  - D (A and B together) – 16 bits
- **Index Registers** -- registers that hold all or part of an address
  - X and Y – 16 bits
- **Stack Pointer** -- register that holds an address for a stack (last-in-first-out memory organization)
  - SP – 16 bits
- **Program Counter** -- register that holds the address of the next instruction
  - PC – 16 bits
- **Condition Code Register** -- register that holds one-bit flags
  - CC – 8 bits

How Do We Store Stuff in a Computer?

- **2-D arrays** are easy to build in VLSI.
- Hence they are used for cheap, fast storage.
  - the larger, the slower.
  - later we will see a hierarchy of memory.
- **Memory** is a collection of register arranged sequentially and referred to by a four-digit hexadecimal address
  - Now we will think of memory as a 1-D array.
  - though actually 2-D grid.

Byte-Addressable Memory

- **Modern computers** can address each byte of the memory.
- We think of memory as a sequence of bytes.
- Each memory location has an address
  - like a post office box address
  - starting from 00 or 0000 hex
- and contents
  - the stuff we store.

Big and Little Endian

- **How can we store a multiple-byte number** in memory?
- **Most significant byte first** - big endian
  - Motorola 68HC11, Sun, Macintosh, PowerPC, MIPS
- **Least significant byte first** - little endian
  - PCs, Alpha, 'x86: Intel, AMD
- **MIPs machines use both.**
- **Byte ordering can be a problem**
  - especially when connecting different processors
Byte Ordering Illustration

- Example: Store 4-byte BCD number 12345678 starting in memory location $0400.

- Big endian — Least significant byte has highest address.
  - 0400: 12
  - 0401: 34
  - 0402: 56
  - 0403: 78

- Little endian — Least significant byte has lowest address.
  - 0400: 78
  - 0401: 12
  - 0402: 34
  - 0403: 56

Memory Organization-Bit Ordering

- Which end of the word is labeled bit 0?
  - Little endian — Bit 0 is the least significant bit (rightmost bit).
    - 0400: 12, 0001,0010
    - 0401: 34, 0011,0100
    - 0402: 56, 0101,0110
    - 0403: 78, 0111,1000
  - Big endian — Bit 0 is the most significant bit (leftmost bit).
    - Mem. Loc. Big Endian Little Endian Big Endian
    - 0400 12 0001,0010 0100,1000
    - 0401 34 0011,0100 0010,1100
    - 0402 56 0101,0110 0101,0110
    - 0403 78 0111,1000 0001,1110

Memory Org. - Granularity

- Granularity = the size of one memory word
  - Almost all modern processors are "byte addressable"
    - one memory word = one byte
    - simplifies access to character data
    - 68HC11 is byte-addressable
  - Proc. “data word” can be larger than one mem. word
    - Most processors are byte addressable
    - most procs can compute with 16, 32, or 64-bit data
    - 68HC11 is an 8-bit processor with LIMITED 16 bit operations
      - Accumulator D is implemented using A & B

6811 Memory Model

- 8-bits (1 byte)/word= Byte-addressable
- 16-bit address => 2^16
  (64k) bytes maximum
- Big-endian byte-ordering
- Little-endian bit-ordering

Figure 2-2. Memory Model
Instruction Formats

- **Fixed Length Instruction Formats**
  - All instructions are the same length (32 bits)
  - Microprocessor data bus must be same length
  - Fast & easy fetch

- **Variable Length Instruction Formats**
  - Instruction length is variable (1-9 bytes)
  - Compact assembly language
  - Multiple cycle fetch

- **68HC11 Instruction Format**
  - Variable length instructions (1-5 bytes)
  - 8-bit data bus – 1 byte fetched per cycle

Instruction Format

- **Machine Language (ML)** =
  - bit-patterns that the proc. interprets as instructions
  - Each instruction contains
    - Opcode
    - Operand specifiers (e.g. value, address, reg. number)

- **Example:**
  - “Load byte @ address C200 into accumulator A”
  - B6C200
    - recall: 1 hex digit = 4 bits = 1/2 Byte
    - This instruction = 6 hex digits = 3 bytes

**Assembly Language (AL)** =

- A more easily readable representation of ML
- Uses a Mnemonic name for the_OPCODE
- Example: “Load byte @ address C200 into Accum. “A”
  - B6C200 = LDAA C200

- Assembler = a program to translate AL to ML
  - e.g. assembler translates “LDAA” into “B6”
  - ≥ a one-to-one mapping from AL to ML instructions
  - Assembler = simpler version of a compiler

Instructions Stored in Memory

- **Storage of: B6C200**
  - 68HC11 instructions range from one to five bytes
  - This one happens to be three bytes

- **Figure 2-3** Example of an instruction code—Load Accumulator A.
Microprocessor Model
- Lists those registers *(in the microprocessor)* involved in the instruction
  - To begin with we will need:
    - **Accumulators**: general purpose 8-bit data registers
    - **Program Counter**: 16-bit register containing address of next data to be read from memory.
    - **Instruction Register**: 24-bit register to hold instruction and operand address read from memory.
      This one is transparent: its operation is invisible to the programmer.

68HC11 Fetch/Execute Cycle - Details
- Example: 3-Byte Instruction
  - **Fetch Phase**
    - Fetch Opcode,
    - Decode Opcode (need 2 more bytes) & Increment PC
    - Fetch 2nd byte & Increment PC
    - Fetch 3rd byte & Increment PC
  - **Execute Phase**
    - Execute the instruction

3-Byte Instruction Cycle - More Details
- **Beginning of Instruction:**
  - Contents of A, B, and IR are “leftovers”
  - PC contains address of 1st byte of instruction

3-Byte Instruction Cycle - More Details (cont.)
- **Tick 1 of Fetch Phase:**
  - Send address of 1st byte and “read” signal to mem
  - Put byte returned by mem into left-hand byte of IR
  - Increment the PC
  - Decode Opcode “B6” -> need 2 more bytes
    - Determine if fetching should continue by examining the left byte of IR

Figure 2-4: The microprocessor model.

Figure 2-7: First byte of instruction has been fetched.
3-Byte Instruction Cycle - More Details

- Tick 2 of Fetch Phase:
  - Send address of 2nd byte and "read" signal to mem
  - Put byte returned by mem into middle byte of IR
  - Increment the PC

![Diagram](image1)

Figure 2-9: The second byte of the instruction has been fetched

- Tick 3 of Fetch Phase:
  - Send address of 3rd byte and "read" signal to mem
  - Put byte returned by mem into right-hand byte of IR
  - Increment the PC

![Diagram](image2)

Figure 2-11: The third byte of the instruction has been fetched

3-Byte Instruction Cycle - More Details

- Tick 4 - Execute Phase
  - Send right-2 bytes of IR and "read" signal to mem
  - Put byte returned by mem into accumulator “A”

![Diagram](image3)

Figure 2-13: The data number has been put into the accumulator.

68HC11 Instruction Execution

- Follows 2-step instruction sequencing
  - Fetch
  - Execute

- Each step can take multiple ticks (clock cycles)
- Up to 41 ticks for a single instruction
- Up to 5 bytes for a single instruction
- Numbers for bytes & ticks in Appendix A of Miller

- Reading Assignment: §2.2, pp. 66-77, §2.3, Miller
  - Topic: instructions and addressing modes